Object-Oriented Programming

Design Pattern Command

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

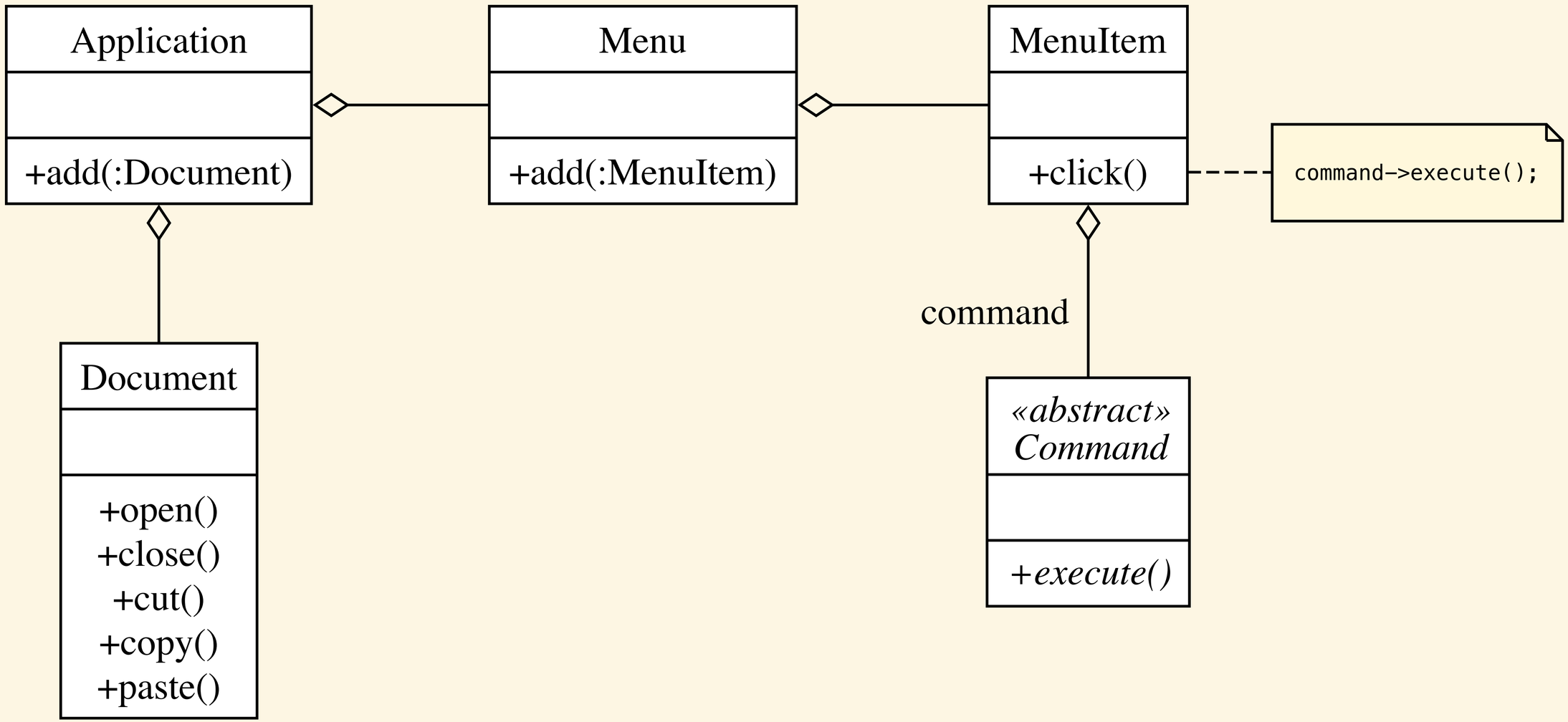
Command

Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations

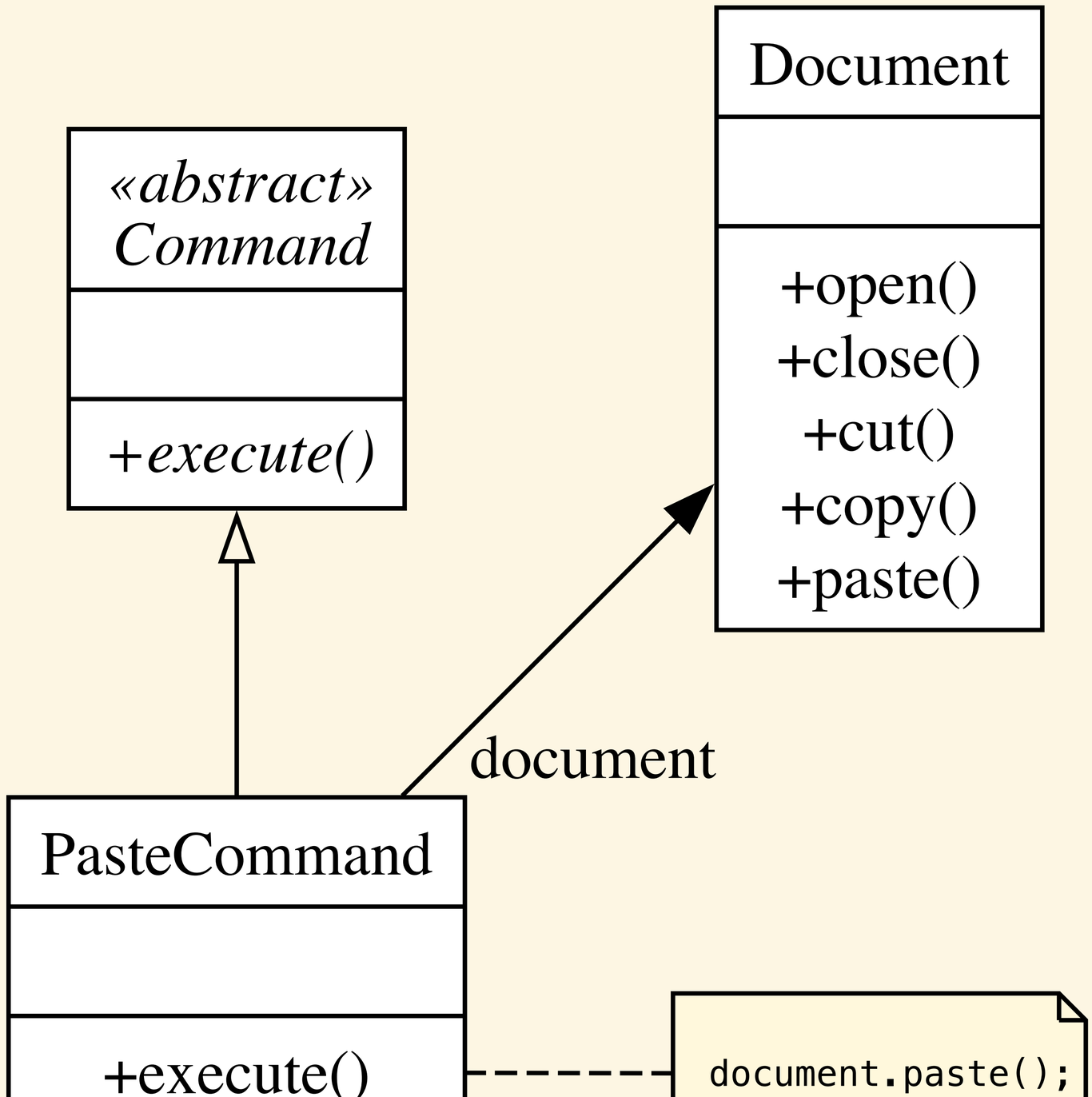
- Behavioral Pattern

- AKA: Action, Transaction

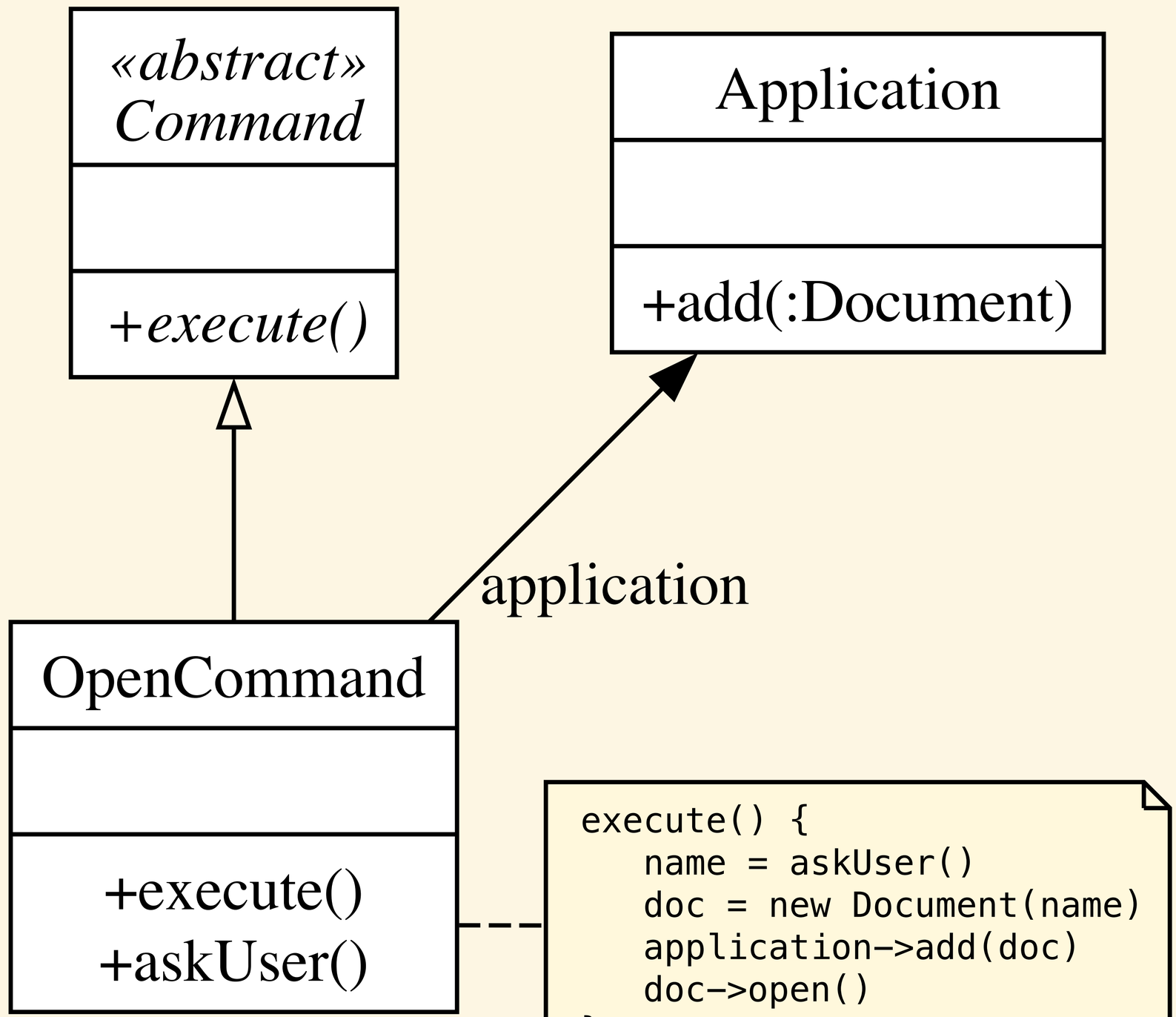
Command: Motivation



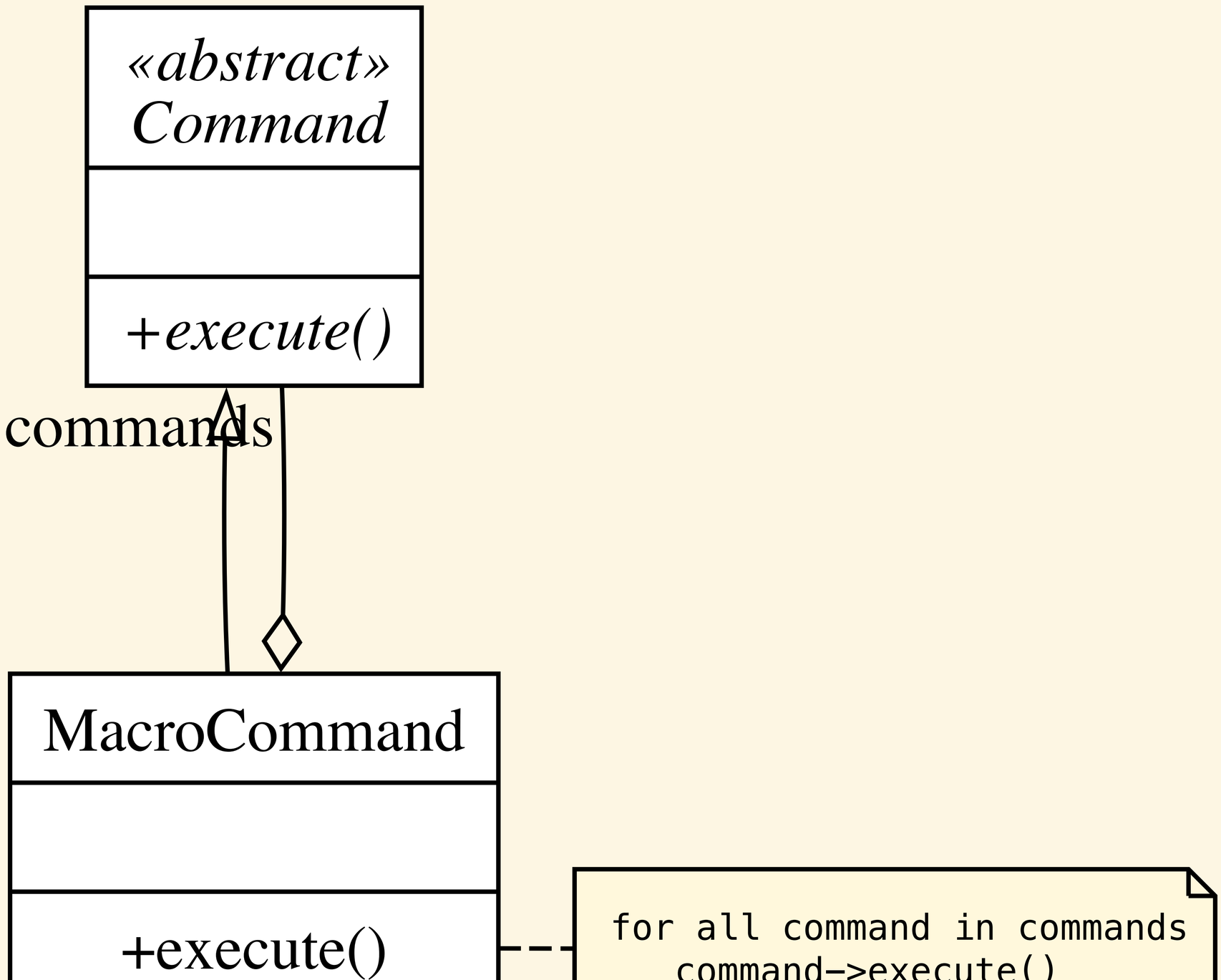
Paste Command



Open Command



Macro Command



Command: Applicability

- Use the command pattern when you want to:

  Parameterize objects by an action to perform, as MenuItem objects did above.

  Commands are an object-oriented replacement for callbacks.

  Specify, queue, and execute requests at different times

Command: Applicability

- Support undo

  `execute()` stores state for reversal

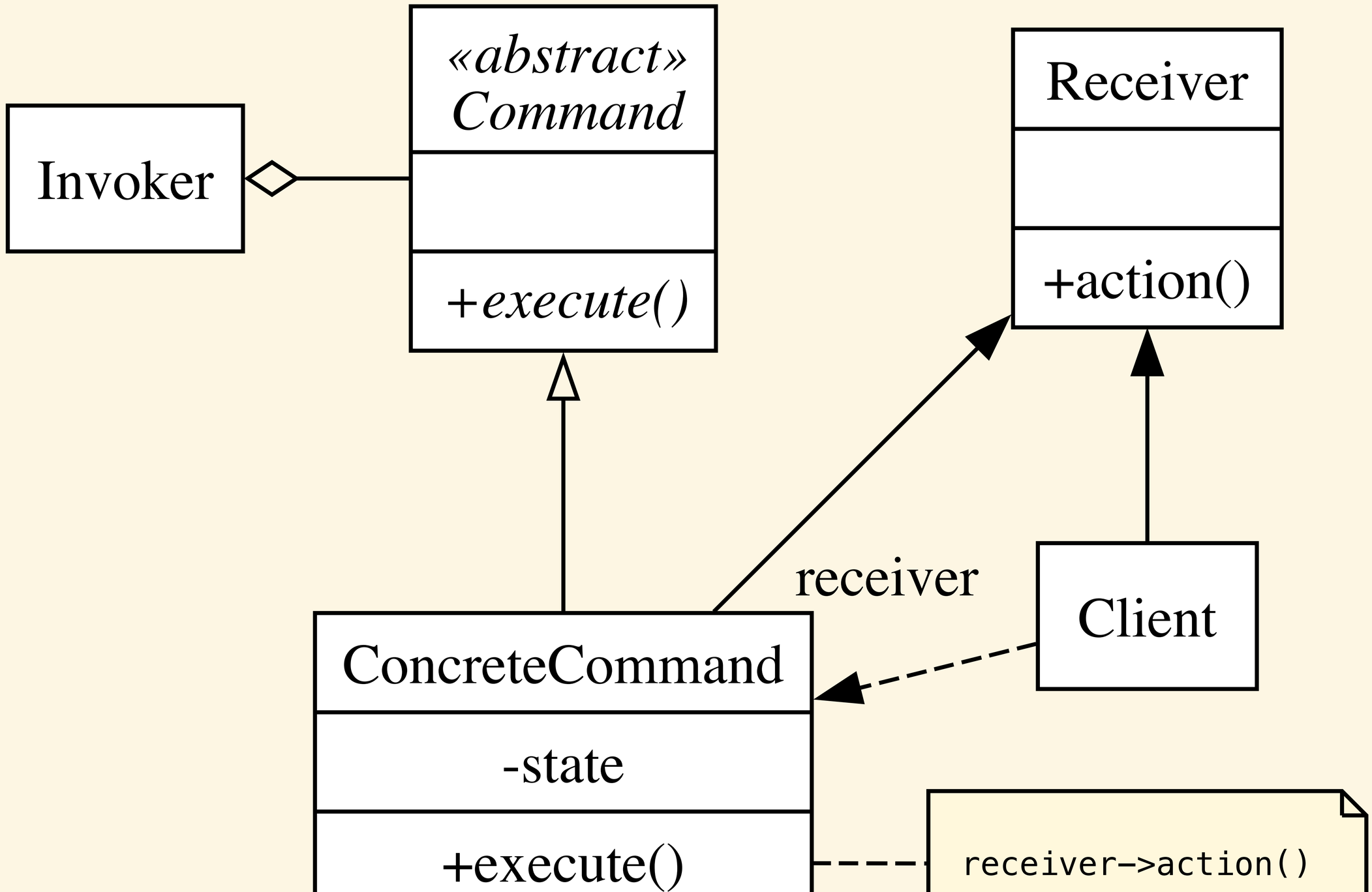
  `unexecute()` reverses the effects of a previous `execute()`

  Executed commands are stored in a history list, with unlimited undo and redo

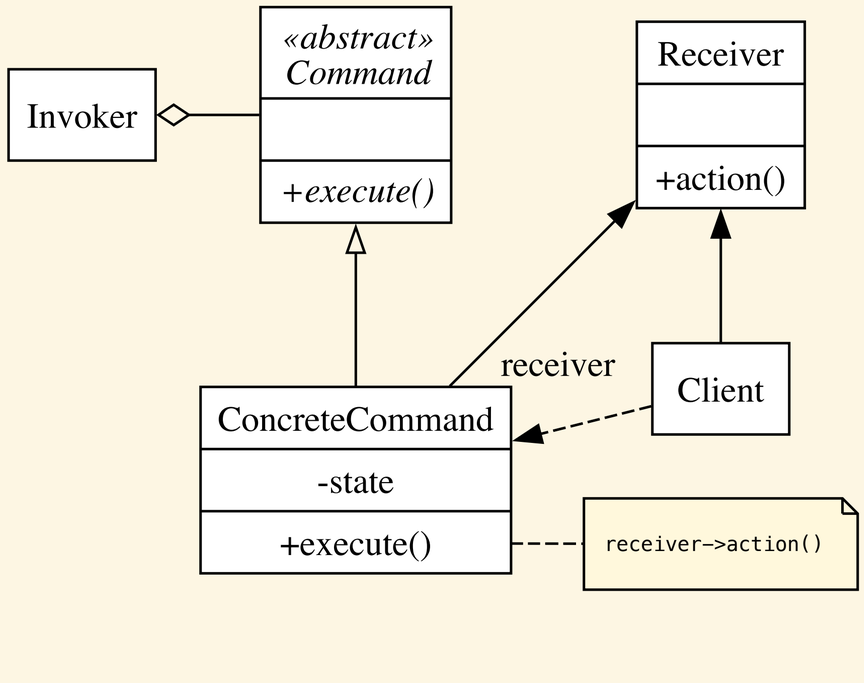
- Support logging changes to reapply in case of a system crash

- Structure a system around high-level operations built on primitive operations and support transactions

Command: Structure



Command: Participants



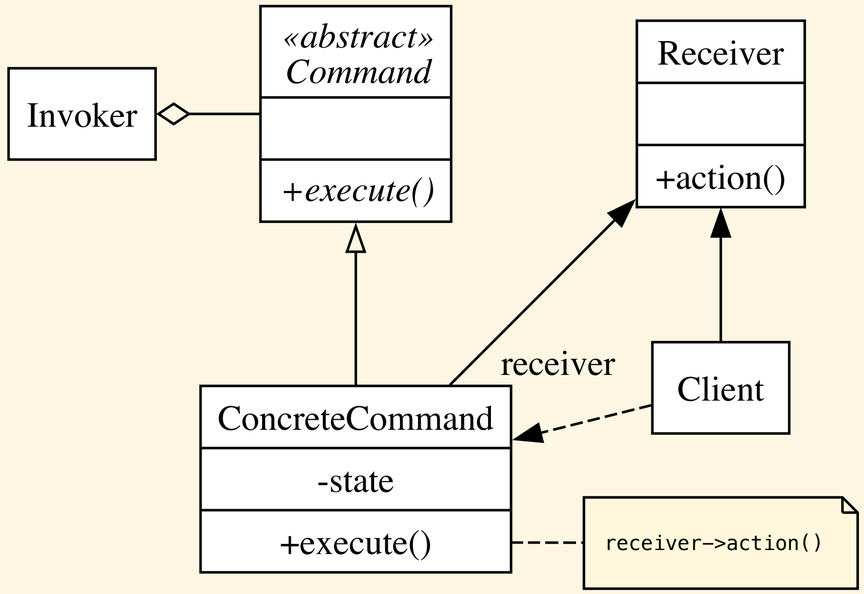
- Command - Declares an interface for executing an operation.

- ConcreteCommand (e.g., PasteCommand, OpenCommand)

  Defines a binding between a Receiver object and an action

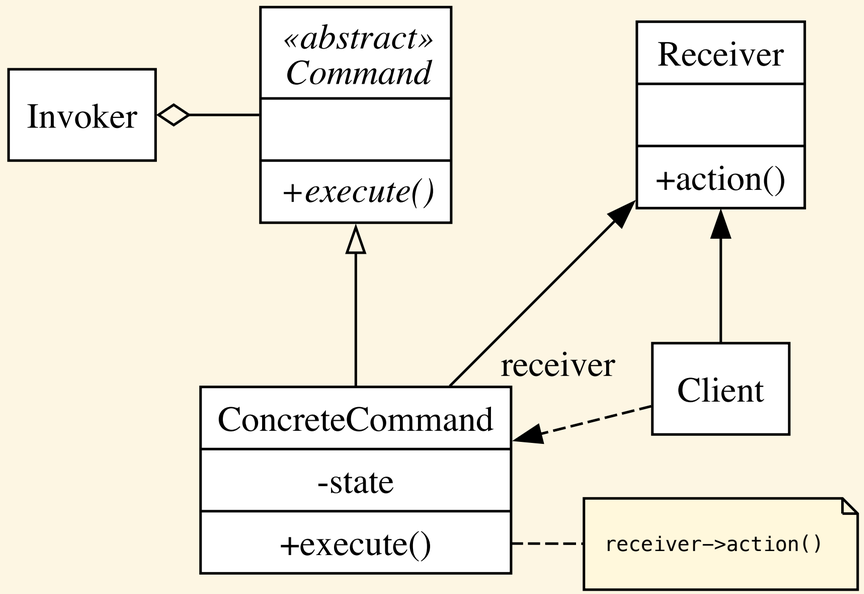
  Implements `execute()` by invoking the corresponding operation(s) on Receiver

Command: Participants



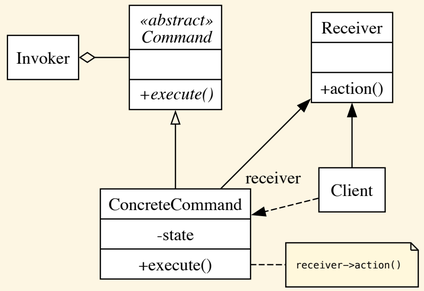
- Client (e.g., Application) Creates a ConcreteCommand object and sets its receiver
- Invoker (e.g., MenuItem) Asks the command to carry out the request
- Receiver (e.g., Document, Application) Knows how to perform the operations associated with carrying out a request

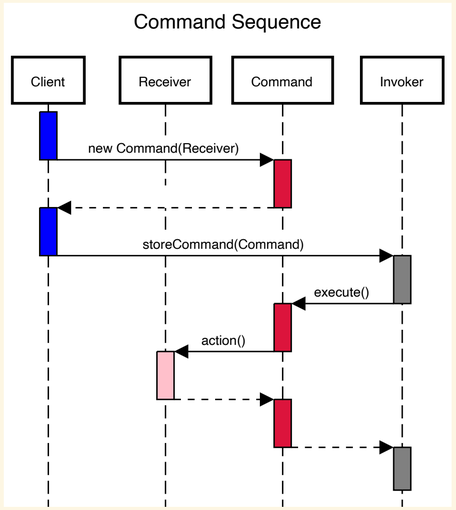
Command: Collaborations



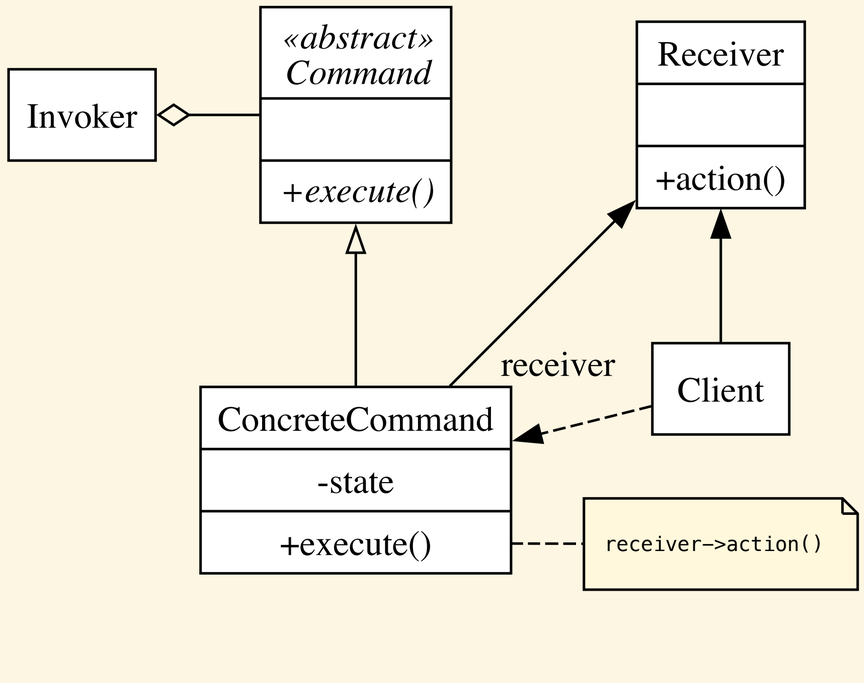
- The client creates a ConcreteCommand object and specifies its receiver
- An Invoker object stores the ConcreteCommand object.
- The invoker issues a request by calling `execute()` on the command. When commands are undoable, ConcreteCommand stores the state for undoing the command before invoking Execute.
- The ConcreteCommand object invokes operations on its receiver to carry out the request.

Flow of Control



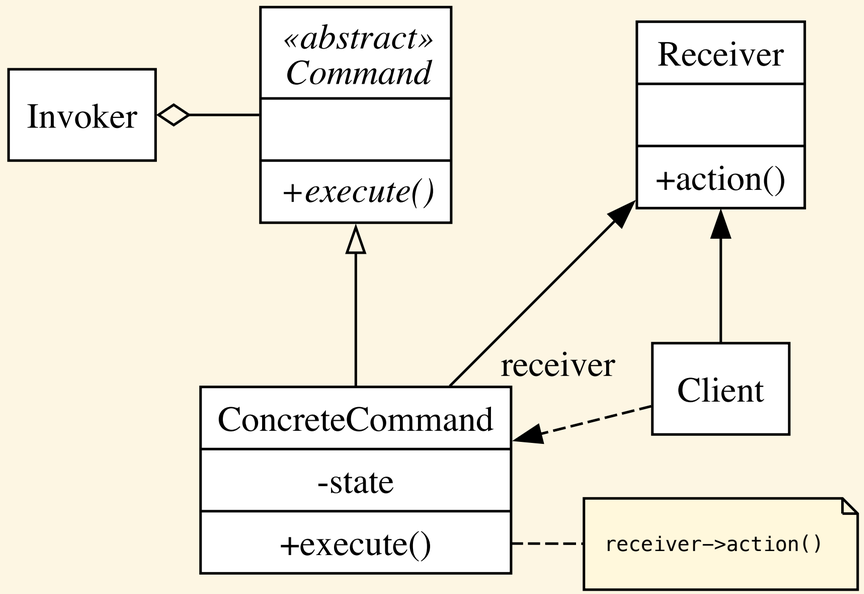


Command: Consequences



- Command decouples the object that invokes the operation from the one that knows how to perform it
- Commands are first-class objects and can be manipulated and extended like any other object.
- Multiple commands can be assembled into a composite command (e.g., MacroCommand) and are instances of the Composite pattern
- It is easy to add new Commands because you don’t have to change existing classes.

Implementation



- How intelligent should a command be?
- Supporting undo and redo
- Avoiding error accumulation in the undo
- Using C++ templates to avoid creating Command subclasses

Related Patterns

- *Composite*

  Used to implement MacroCommands

- *Memento*

  Keep state the command requires for an undo

- *Prototype*

  Command is copied before being placed on a history list and acts as a Prototype